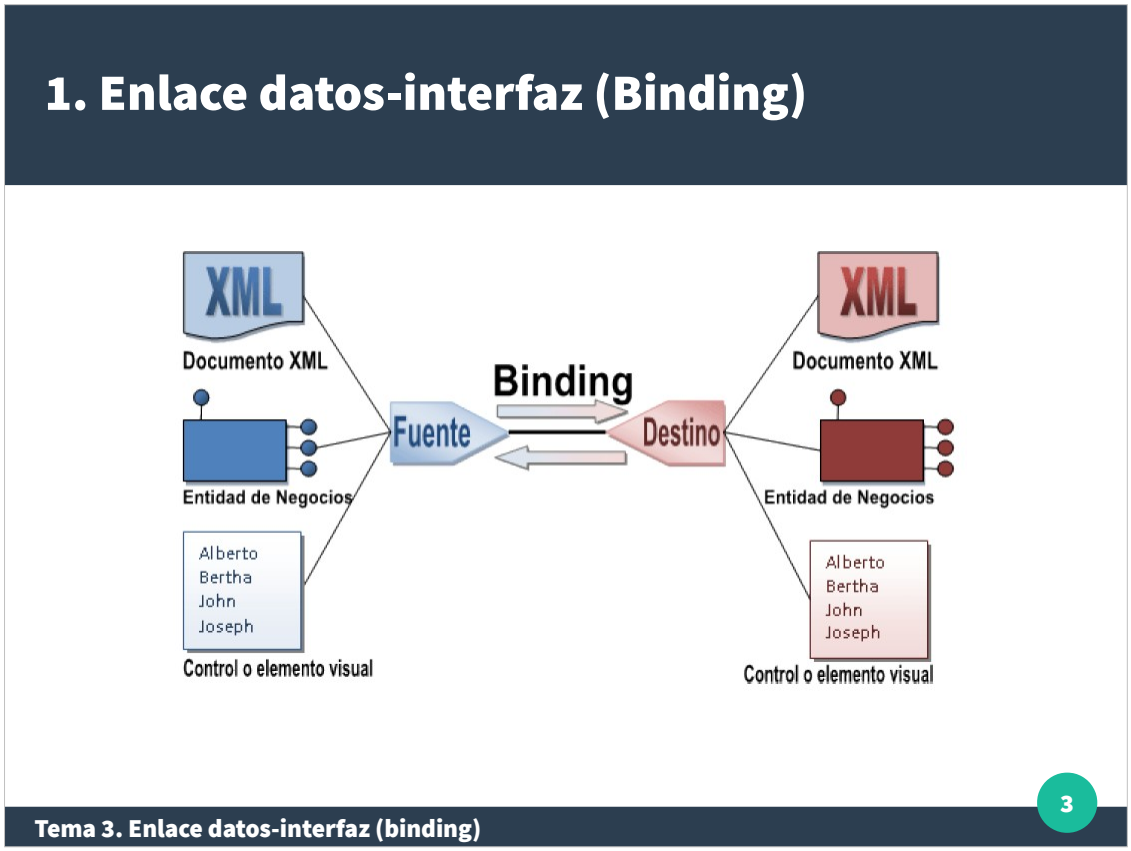
Tema 3. Enlace datos-interfaz (binding)

Desarrollo de Interfaces
DAM – IES Doctor Balmis

Javier Catalá

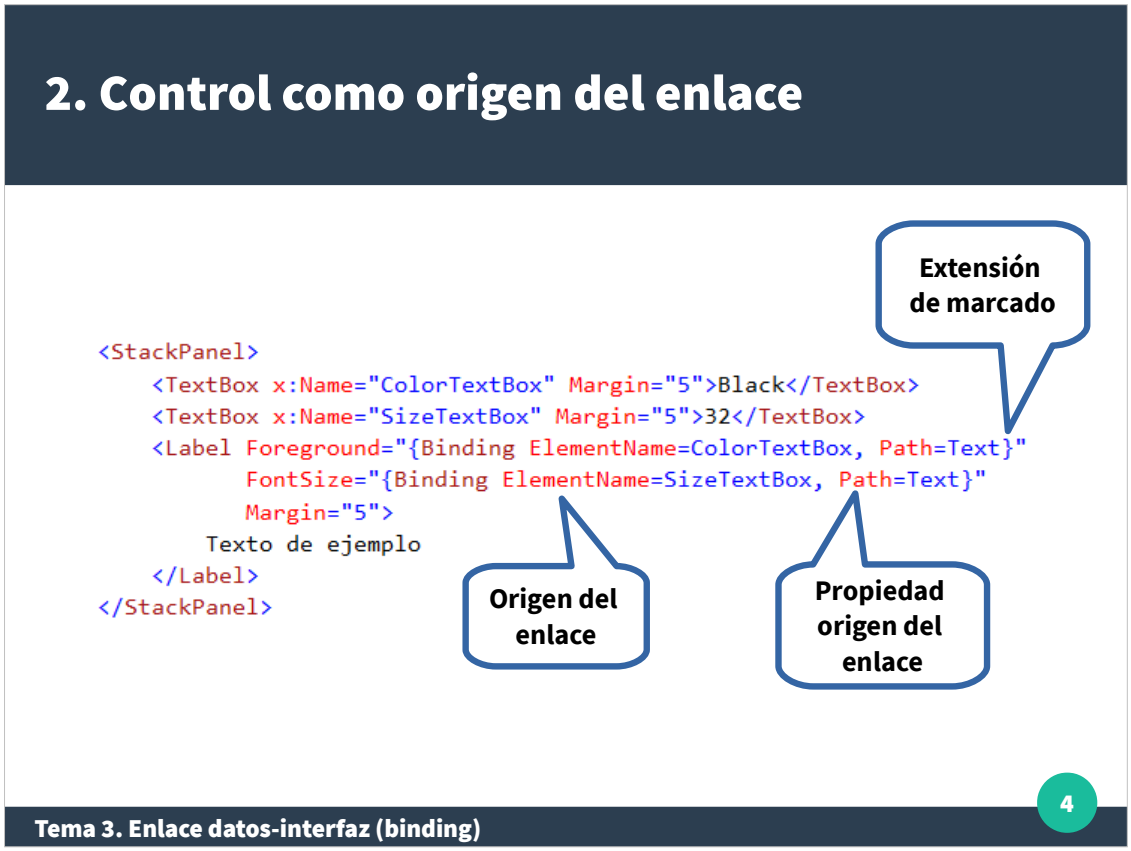
INDICE



El mecanismo conocido como *binding* permite enlazar una propiedad de un elemento fuente (también llamado origen) con otra propiedad de un elemento destino.

Los elementos origen y destino, como se aprecia en la imagen, pueden ser cualquier objeto de nuestra aplicación (entidad de negocio o control) o incluso un documento XML. Lo más habitual será que el destino del enlace sea un control de nuestra interfaz.

Aunque en principio el binding está pensado para que el valor de la propiedad del objeto origen se traslade a la propiedad del objeto destino, este mecanismo también permite que el valor en el destino se traslade al origen.

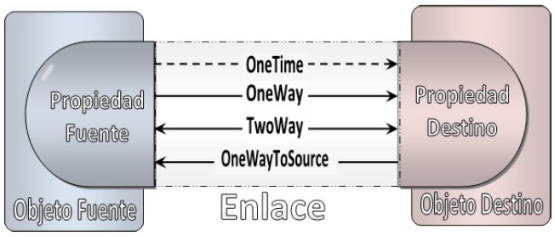


Una de las opciones que nos ofrece este mecanismo es al de utilizar controles de la interfaz como origen y destino del *binding*.

Como se puede apreciar en la imagen, el enlace se especifica en la propiedad destino del control destino del enlace. La expresión de enlace se indica utilizando la extensión de marcado ({ }) *Binding*.

En esta extensión de marcado indicaremos el control origen del enlace mediante la propiedad *ElementName*, y la propiedad origen mediante la propiedad *Path*. *Path* es la propiedad por defecto de la extensión de marcado *Binding*, por lo que se puede incluir su valor sin hacer referencia a ella.

3. Modo del enlace




```
<StackPanel>
    <TextBox x:Name="ColorTextBox" Margin="5">Black</TextBox>
    <TextBox x:Name="SizeTextBox" Margin="5">32</TextBox>
    <Label Foreground="{Binding ElementName=ColorTextBox, Path=Text}"
           FontSize="{Binding ElementName=SizeTextBox, Path=Text, Mode=TwoWay}"
           x:Name="TextoLabel" Margin="5">
        Texto de ejemplo
    </Label>
</StackPanel>
```

Enlace bidireccional

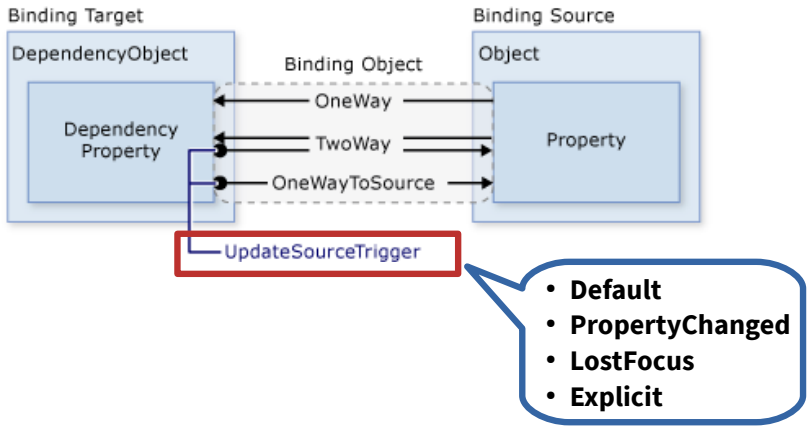
Tema 3. Enlace datos-interfaz (binding) 5

Otra característica del enlace que podemos indicar al definirlo es el modo (propiedad *Mode*). Esta propiedad puede tener los siguientes valores:

- *OneWay*: se traslada únicamente el valor de la propiedad origen a la propiedad destino.
- *TwoWay:* se trasladan también los cambios en la propiedad destino a la propiedad origen.
- *OneWayToSource:* solo se trasladan los cambios de la propiedad destino a la propiedad origen.
- *OneTime:* es como un enlace *OneWay*, pero solo se traslada el valor una vez.

El valor por defecto del modo del enlace dependerá de la propiedad que estemos enlazando en el control origen. Lo más habitual es que el modo por defecto sea *TwoWay* o *OneWay*.

3. Modo del enlace

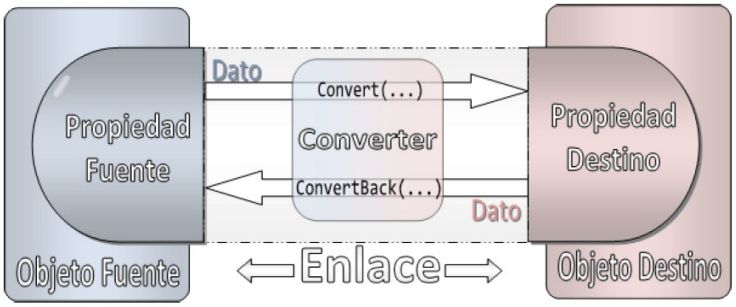



```
<StackPanel>
    <TextBox x:Name="ColorTextBox" Margin="5"
             Text="{Binding ElementName=TextoLabel, Path=Foreground, Mode=TwoWay,UpdateSourceTrigger=LostFocus}"/>
    <TextBox x:Name="SizeTextBox" Margin="5"
             Text="{Binding ElementName=TextoLabel, Path=FontSize, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
    <Label x:Name="TextoLabel" Foreground="Black" FontSize="18">Texto de ejemplo</Label>
</StackPanel>
```


En los enlaces en los que los cambios en el destino se propagan al origen (*TwoWay* y *OneWayToSource*) se puede utilizar la propiedad *UpdateSourceTrigger* del *Binding* para controlar cuando se traslada el valor:

- *PropertyChanged:* se actualiza el origen en cuanto cambia el valor en el destino.
- *LostFocus:* no se actualiza el origen hasta que el destino pierda el foco.
- *Explicit:* la actualización se tiene que hacer de forma explícita, desde el código.
- *Default*: el valor por defecto, que dependerá de las propiedades enlazadas.

4. Convertidores en el enlace



Otra posibilidad que nos ofrecen los enlaces es la de aplicar una conversión en el valor que se traslada, antes de que éste llegue a su destino. Para ello es necesario definir una clase llamada convertidor, que contendrá los métodos para realizar la conversión:

- *Convert*: realizará la conversión cuando el valor se traslade del origen al destino.
- *ConvertBack*: realizará la conversión cuando el valor se traslade del destino al origen (en el caso de enlaces *TwoWay*).

Como se puede apreciar en el ejemplo, la clase convertidora debe implementar la interfaz *IvalueConverter*.

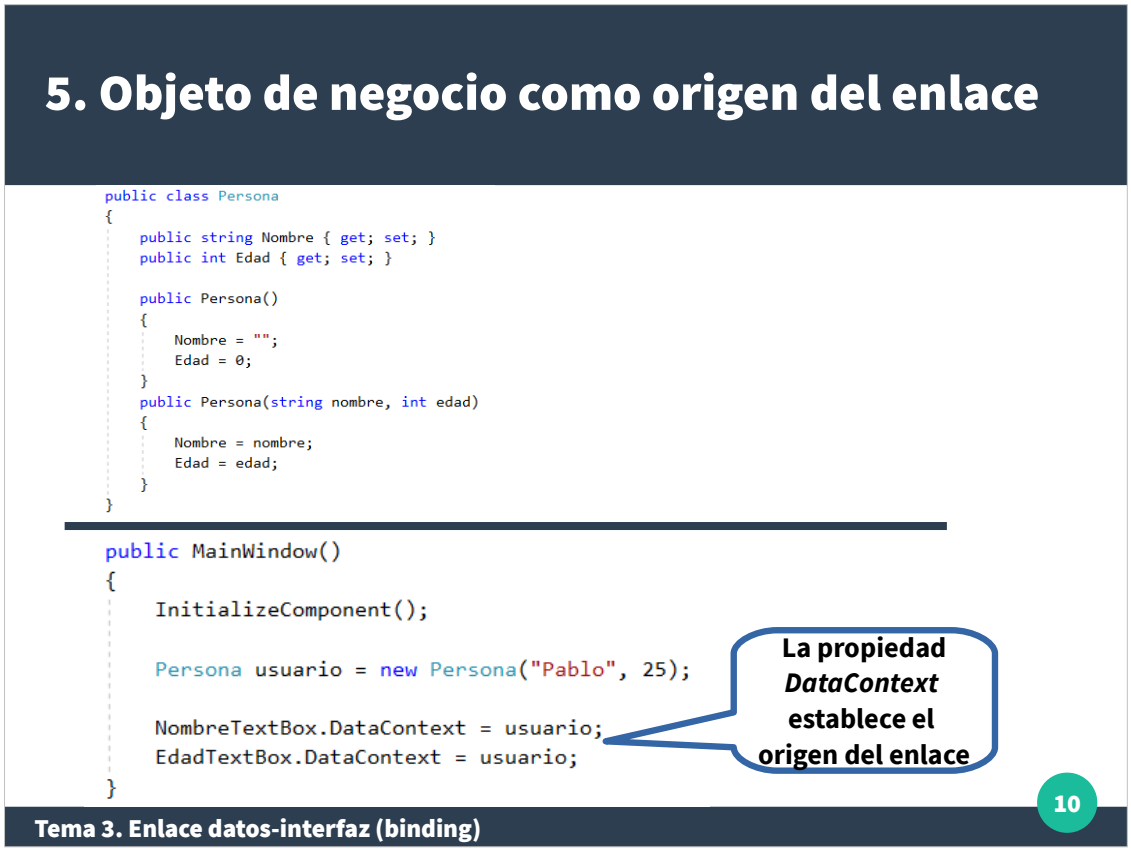
En el parámetro *value* (de tipo *object*) tenemos disponible el valor previo a la conversión. Tras aplicar a este valor la conversión correspondiente, devolveremos el valor convertido.

En estos momentos, el resto de parámetros del conversor no los utilizaremos.

Para utilizar el conversor, en primer lugar tenemos que crear una instancia (objeto) de la clase conversor que hemos creado. Para poder hacer referencia a esta instancia desde XAML, la crearemos directamente en este lenguaje.

En la propiedad *Resources* de la ventana, haremos referencia a la clase creada (*YesNoToBooleanConverter* en el ejemplo) y daremos un nombre a la instancia con la propiedad *x:Key* (en el ejemplo, *conversor*).

A continuación, deberemos utilizar la propiedad *Converter* en el *Binding* haciendo referencia a la instancia creada en los recursos de la ventana.



Además de poder utilizar como origen del enlace un control de la interfaz, también podemos usar como origen un objeto de negocio. Para hacer referencia al objeto que se utilizará como origen de los enlaces que se hagan en un control se utiliza la propiedad *DataContext*.

Cuando utilizamos como origen del enlace un control, el *ElementName* se aplica únicamente a un enlace. Sin embargo, el objeto definido en el *DataContext* será el origen por defecto para todos los enlaces del control.

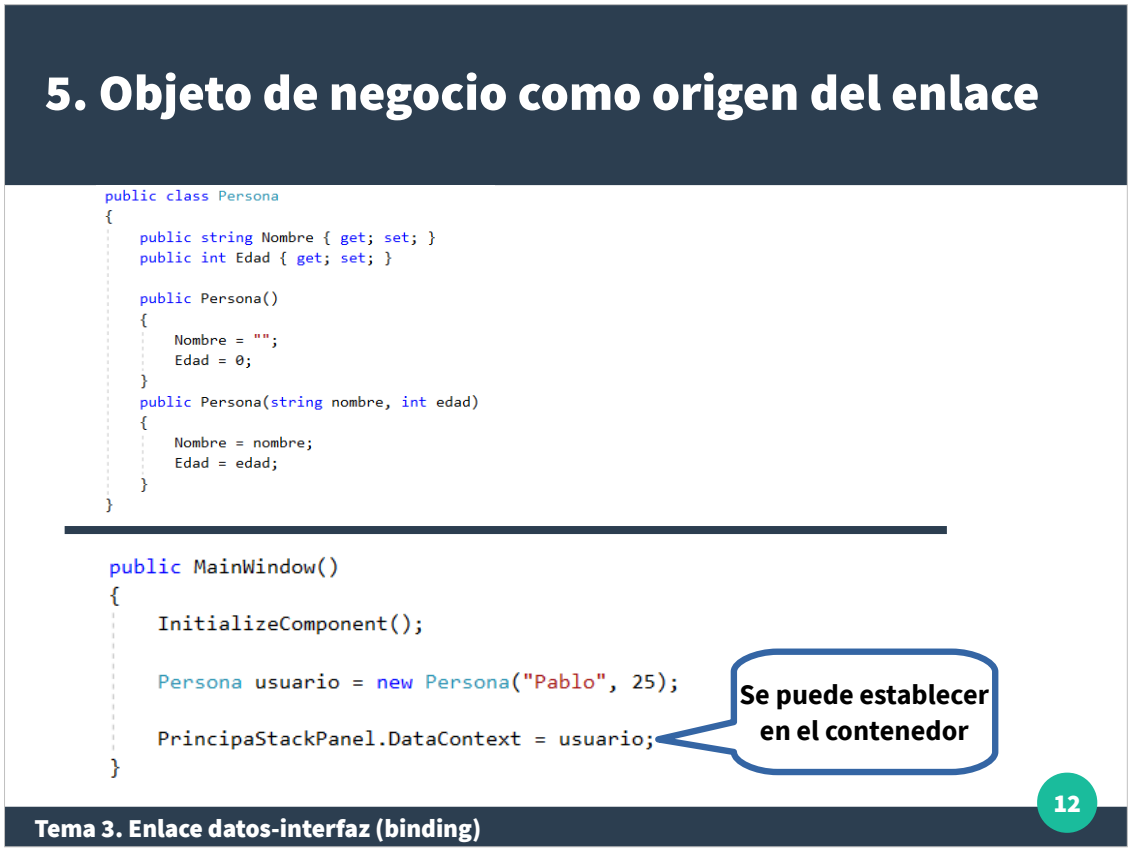
En el ejemplo se define una clase *Persona*. En el constructor de la ventana, se instancia un nuevo objeto persona, y se asigna a la propiedad *DataContext* de los dos *TextBox*.

5. Objeto de negocio como origen del enlace

```
<StackPanel x:Name="PrincipalStackPanel">
    <Label>Nombre:</Label>
    <TextBox x:Name="NombreTextBox" Text="{Binding Path=Nombre}" Margin="5,0,5,5"></TextBox>
    <Label>Edad:</Label>
    <TextBox x:Name="EdadTextBox" Text="{Binding Path=Edad}" Margin="5,0,5,5"></TextBox>
    <Button x:Name="boton">Consultar</Button>
</StackPanel>
```

Propiedad enlazada del objeto origen del enlace

Una vez establecido el *DataContext* de un control, se puede establecer el enlace haciendo referencia únicamente a la propiedad adecuada del objeto origen.



La propiedad *DataContext* se puede establecer en un panel, de forma que todos los controles contenidos en el panel tendrán el mismo objeto origen para los enlaces.

A pesar de definir el *DataContext* para un contenedor, podemos establecer un *DataContext* diferente para uno de los controles contenidos en él. Siempre prevalecerá el *DataContext* más específico.

Como vimos en el apartado de convertidores, es posible instanciar un objeto directamente en XAML. Podemos utilizar está característica para instanciar el objeto que hará de origen del enlace.

En el ejemplo, se aprecia como se instancia un nuevo objeto de la clase *Persona*, y se utiliza como *DataContext* del contenedor (utilizando la extensión de marcado *StaticResource*).

Es importante destacar que para poder utilizar esta opción es necesario que la clase implemente un constructor sin parámetros.

Otra opción que nos ofrece el mecanismo de binding es la de utilizar los objetos instanciados en XAML como origen de un enlace concreto. Para ello, se utiliza la propiedad *Source* del enlace.

A diferencia de cuando utilizamos el *DataContext*, el objeto origen definido con la propiedad *Source* aplicará solamente al enlace en el que se indique. Además, es posible combinar ambos mecanismos en el mismo control, prevaleciendo siempre el *Source* sobre el *DataContext*.

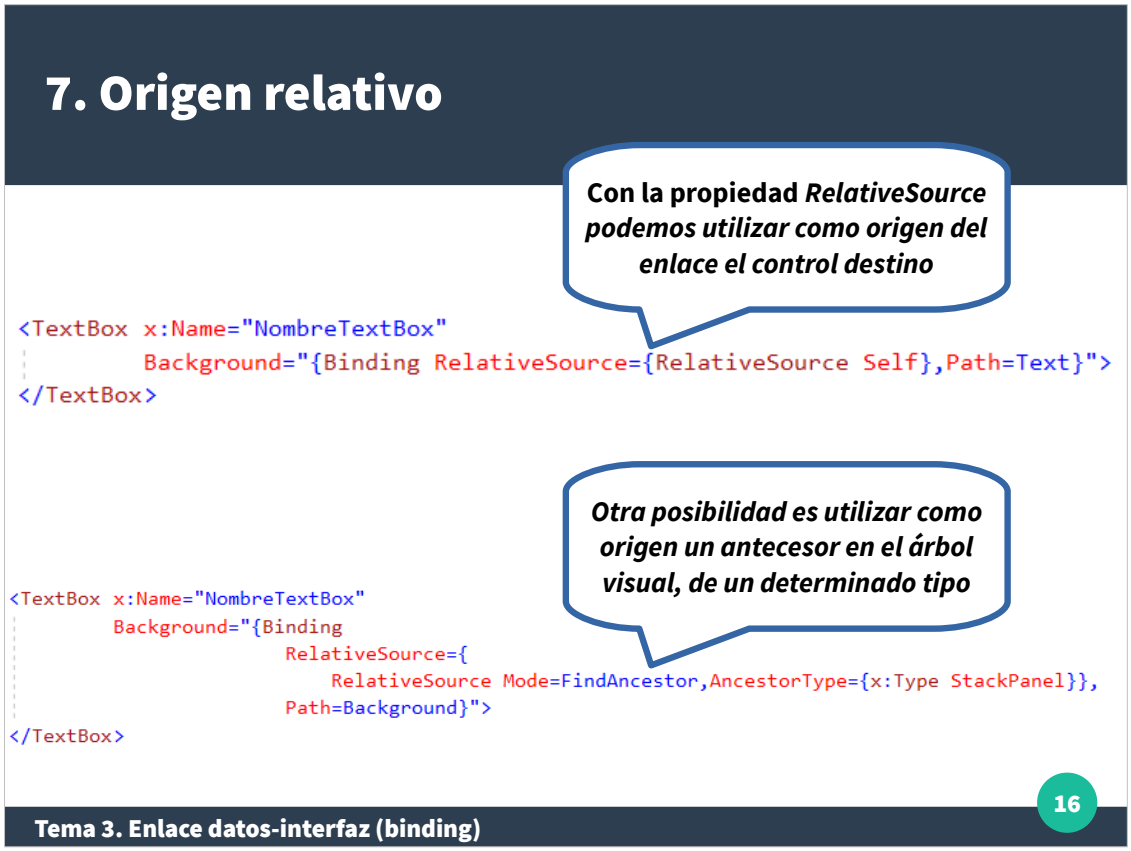
6. Interfaz INotifyPropertyChanged

```
public class PersonaINotify : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string propertyName)
    {
        this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
```

```
private string _nombre;
public string Nombre
{
    get { return this._nombre; }
    set
    {
        if (this._nombre != value)
        {
            this._nombre = value;
            this.NotifyPropertyChanged("Nombre");
        }
    }
}
```


Las clases que se utilizan como origen de los enlaces deben implementar la interfaz *InotifyPropertyChanged*. Esta interfaz permite que los cambios en las propiedades de los objetos sean notificados a los destinos de los enlaces. Si no se implementa, los cambios no se verán reflejados en el destino.

La implementación implica la definición de un evento (*PropertyChanged*) y la implementación de un método que lance el evento (*NotifyPropertyChanged*). Este método será llamado desde los *setters* de las propiedades, en caso de que se se esté modificando el valor de la propiedad.



Otra posibilidad que ofrece el mecanismo de *binding* es la de utilizar un origen relativo para el enlace, mediante la propiedad *RelativeSource*.

Una de las funcionalidades que ofrece el origen relativo es la de indicar que el origen del enlace será el mismo que el destino (primer ejemplo de la diapositiva).

En el segundo ejemplo se utiliza como origen del enlace un antecesor del control destino en el árbol visual. Para utilizar esta opción es necesario indicar el tipo del antecesor, de forma que se tomará como origen el primer antecesor que coincida con el tipo.

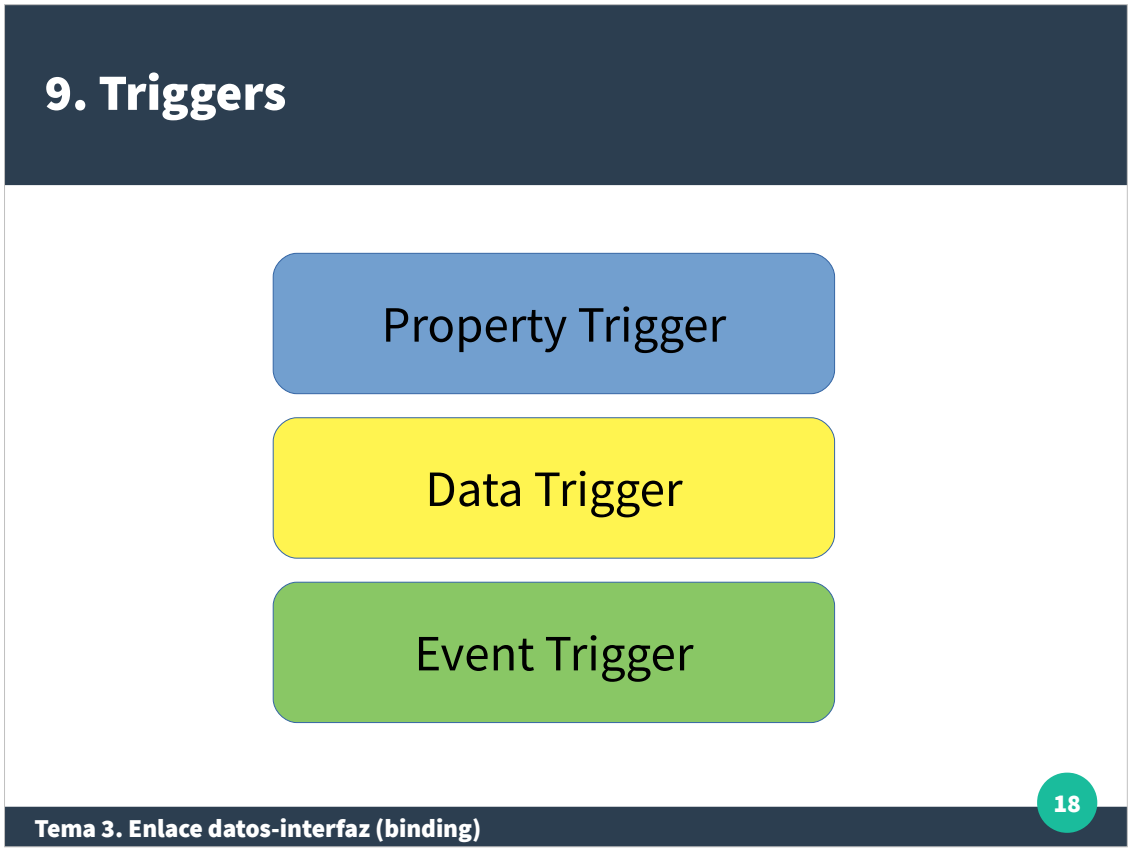
8. Binding desde el código

```
//Creamos un nuevo objeto Binding, indicando el Path (propiedad del origen que enlazamos)
Binding enlace = new Binding("Nombre");

//Configuramos el enlace según sea necesario (Converter, Mode, Source,...)
enlace.Mode = BindingMode.TwoWay;

//Asociamos el enlace creado a la propiedad destino del control destino
NombreTextBox.SetBinding(TextBlock.TextProperty, enlace);
```


Para definir un enlace desde el código trasero se utiliza la clase *Binding*. Al crear un objeto de esta clase indicamos la propiedad que enlazaremos en el origen.

También en este objeto configuraremos las distintas opciones que hemos visto para los enlaces (modo, convertidores, origen…).

Una vez configurado el enlace, lo asociaremos al destino con el método *SetBinding*, indicando el nombre de la propiedad que se enlaza.

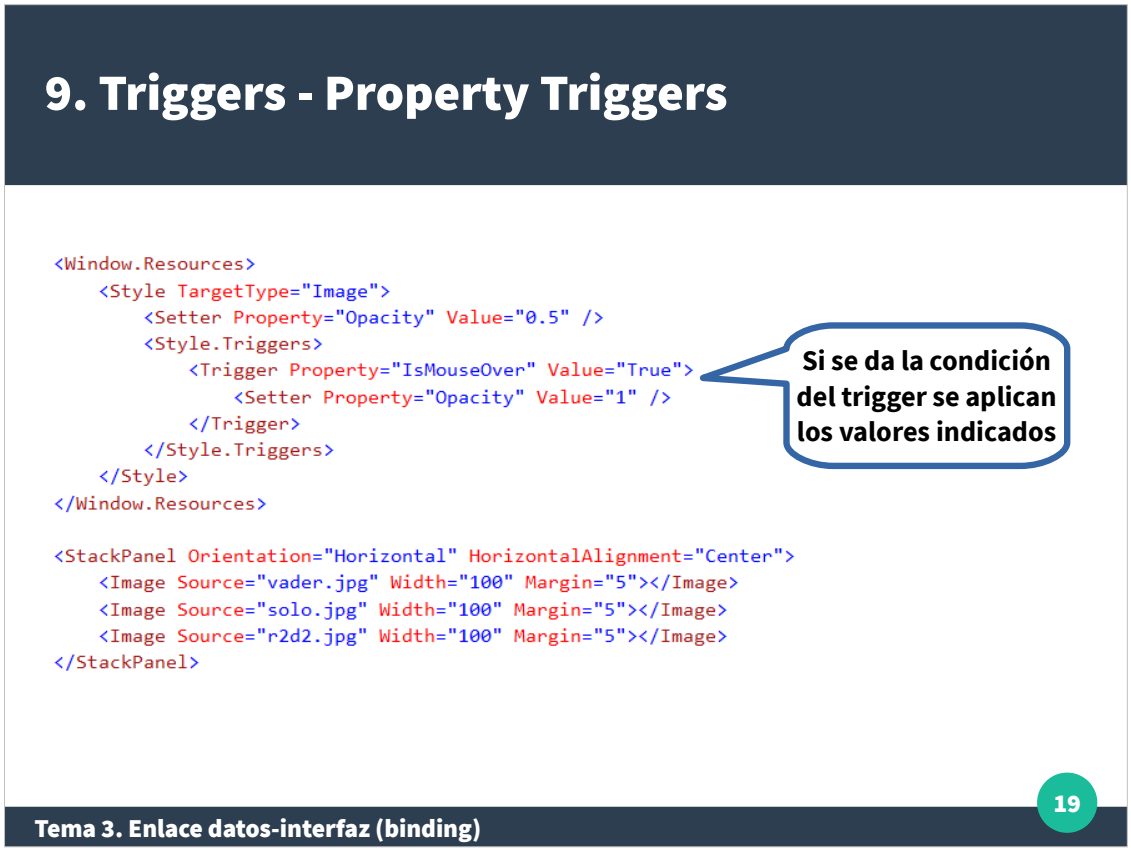


Dentro de este mismo tema vamos a ver otra funcionalidad que nos ofrece WPF con el objetivo de reducir el código trasero asociado a la interfaz de usuario: los *triggers (*disparadores*)*.

Aunque los *triggers* pueden ser definidos en diferentes partes de una aplicación WPF, para estudiar su funcionamiento los utilizaremos únicamente dentro de estilos.

Se pueden distinguir tres tipos de *triggers*:

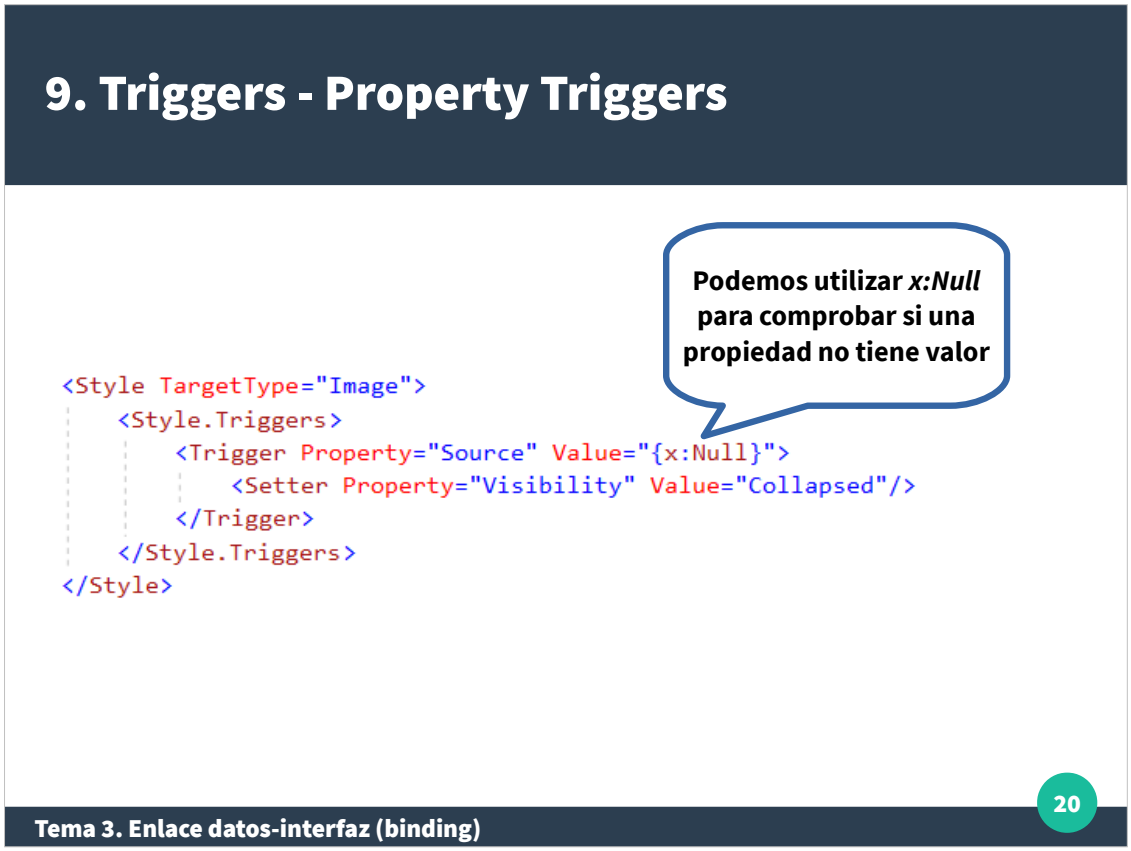
- *Triggers* de propiedad
- *Triggers* de datos
- *Triggers* de evento



Los *triggers* de propiedad permiten que, dentro de un estilo, asignemos un valor a una propiedad solo si se cumple una condición. En este tipo de *triggers* la condición únicamente puede construirse en base a una propiedad del propio control al que se aplica el estilo. Además, solamente se puede utilizar el comparador de igualdad.

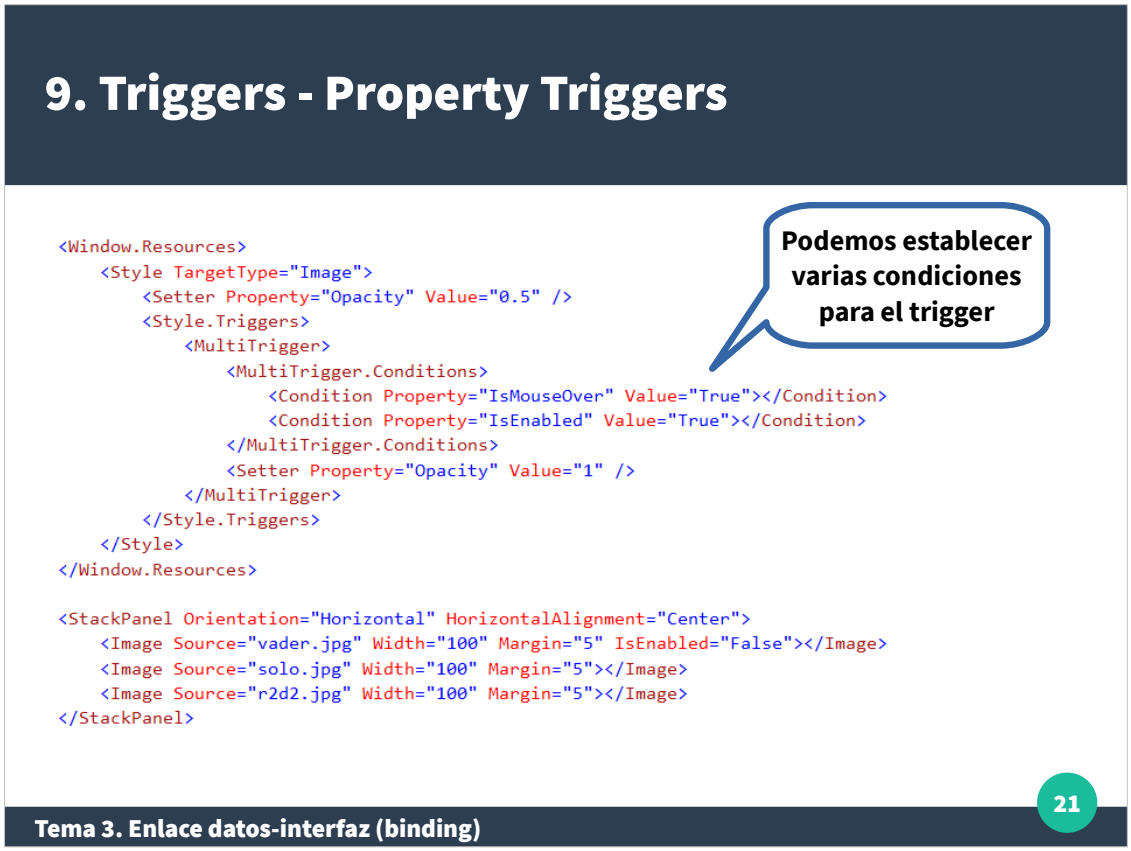
En el ejemplo, se establece como condición del *trigger* que la propiedad *IsMouseOver* de la imagen a la que se está aplicando el estilo tenga el valor *True*. Si se cumple la condición, se dará el valor 1 a la propiedad Opacity.

En cuanto la condición del *trigger* deje de cumplirse, se revertirá el valor de la propiedad.



XAML nos ofrece la constante *x:Null* para poder comprobar en la condición de un *trigger* si una propiedad no tiene valor.

En el ejemplo se comprueba si la propiedad *Source* de la imagen es nula, en cuyo caso a la propiedad *Visibility* se le da el valor *Collapsed*, para que el control no ocupe espacio en la interfaz.



WPF también nos permite establecer más de una condición en el mismo *trigger*. Es lo que se conoce como *multitrigger*. Para que el trigger se dispare se tienen que cumplir todas las condiciones.

En el ejemplo, se establecen dos condiciones en el *multitrigger*: las propiedades *IsMouseOver* y *IsEnabled* deben tener el valor *True*. Cuando se den las dos condiciones al mismo tiempo el valor de la propiedad *Opacity* será uno. En cuanto una de las dos deje de cumplirse se revertirá la propiedad.

9. Triggers - Data Triggers

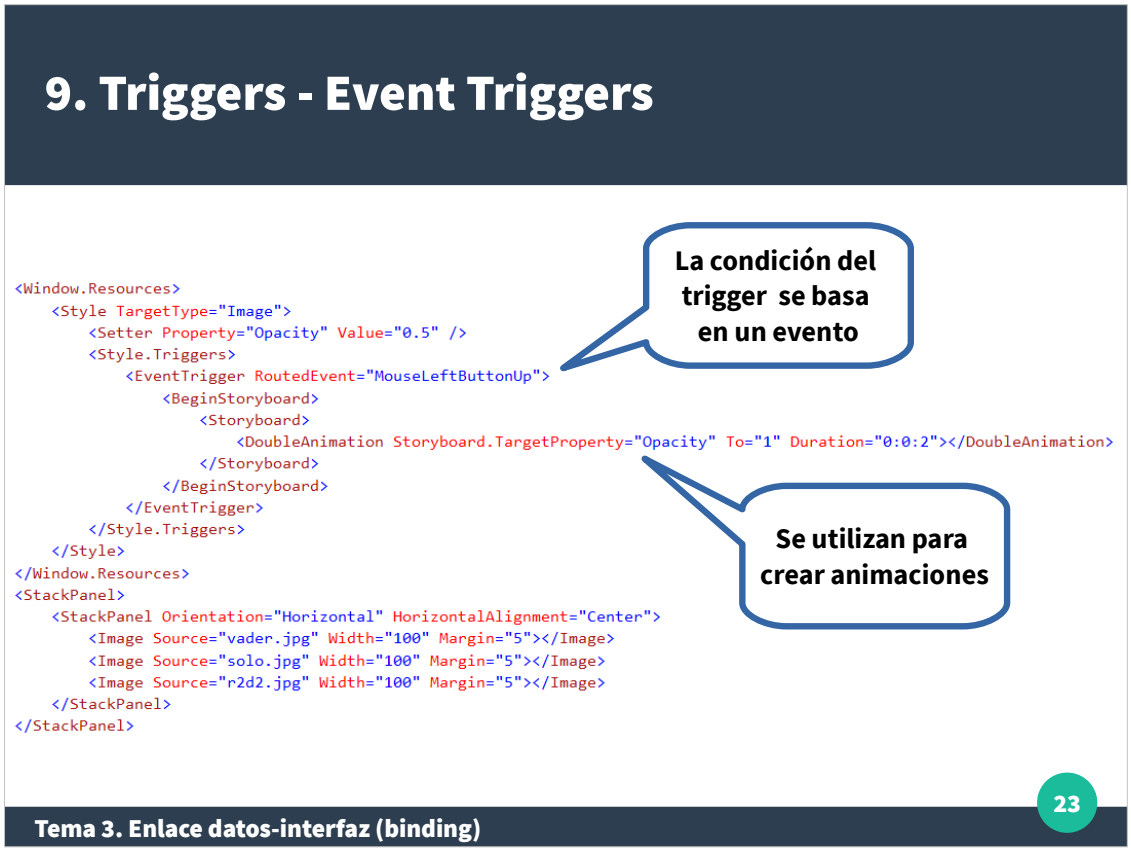
La condición del trigger se basa en un enlace

```
<Window.Resources>
    <Style TargetType="Image">
        <Setter Property="Opacity" Value="0" />
        <Style.Triggers>
            <DataTrigger Binding="{Binding ElementName=MostrarCheckBox, Path=IsChecked}" Value="True">
                <Setter Property="Opacity" Value="1"></Setter>
            </DataTrigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
<StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
        <Image Source="vader.jpg" Width="100" Margin="5"></Image>
        <Image Source="solo.jpg" Width="100" Margin="5"></Image>
        <Image Source="r2d2.jpg" Width="100" Margin="5"></Image>
    </StackPanel>
    <CheckBox x:Name="MostrarCheckBox" Margin="5">Mostrar</CheckBox>
</StackPanel>
```


Los *triggers* de datos tienen una funcionalidad similar a los de propiedad, pero a diferencia de éstos la propiedad que se comprueba en la condición no tiene porque ser del propio control al que se está aplicando el estilo.

En los *triggers* de datos se utiliza una expresión de *binding* para determinar la propiedad que se debe chequear. En esta expresión podremos utilizar todas las características del *binding* que vimos en el apartado anterior.

En el ejemplo, se disparará el *trigger* cuando la propiedad *IsChecked* del control *MostrarCheckBox* sea *True*.




```
<Window.Resources>
    <Style TargetType="Image">
        <Setter Property="Opacity" Value="0.5" />
        <Style.Triggers>
            <EventTrigger RoutedEvent="MouseLeftButtonUp">
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimation Storyboard.TargetProperty="Opacity" To="1" Duration="0:0:2"></DoubleAnimation>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
<StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
        <Image Source="vader.jpg" Width="100" Margin="5"></Image>
        <Image Source="solo.jpg" Width="100" Margin="5"></Image>
        <Image Source="r2d2.jpg" Width="100" Margin="5"></Image>
    </StackPanel>
</StackPanel>
```

El último tipo de disparadores son los *triggers* de evento. En este caso, la condición para que se dispare el *trigger* está asociada a que se produzca un determinado evento sobre el control. En el ejemplo, el *trigger* se disparará cuando se pulse el botón izquierdo del ratón sobre la imagen.

Este tipo de *triggers* se suele utilizar para realizar animaciones. Existen animaciones de diferentes tipos, pero las más sencillas son las que permiten evolucionar el valor de una propiedad desde un valor inicial a un valor final en un intervalo de tiempo.

En el ejemplo, cuando el *trigger* se dispare se irá modificando progresivamente el valor de la opacidad desde el valor que tenga hasta el valor 1, durante 2 segundos.